8 DTVCC Interpretation Layer

The DTVCC Interpretation Layer defines the DTVCC Graphical User Interface. This discussion includes how the caption data coding is to be formatted when encoded and how it is to be interpreted when decoded. While the Caption Data Services Coding Layer (Section 7) identifies how service data bytes are represented, the Interpretation Layer describes how these bytes of data are to be processed. The data bytes for each caption service are interpreted as a unique data stream, independent from the other services.

This section defines all required and optional DTV closed-captioning features. Refer to Section 9 for a list of features required within a minimum DTVCC receiver implementation.

8.1 DTVCC Caption Components

The five major components of DTVCC captioning are Caption Screen, Caption Windows, Caption Pens, Caption Text, and Caption Display Synchronization.

- **Caption Screen**: The Caption Screen is the canvas on which caption windows are displayed.
- **Caption Windows**: The heart of caption definition consists of Caption Windows within which caption text is displayed.
- **Caption Pens**: The Caption Pens component defines styles and attributes for the appearance of the text within the caption windows.
- **Caption Text**: The Caption Text component defines how text is encoded and directed to specific windows.
- **Caption Synchronization**: The Caption Synchronization component controls the flow of interpretation of commands and caption text within the independent service data streams.

The following subsections provide a general overview of these components. These subsections are then followed by a detailed presentation of the DTVCC caption command set.

8.2 Screen Coordinates

A set of coordinates is defined to map a rectangular grid onto the "safe-title" area of the screen. This grid is used as a reference for superimposing captions. This reference is used to specify the position of caption windows.

Receivers which decode the DTV bit stream may have either a 16:9, 4:3 or other display screen aspect ratio. The coordinate-system grid size for a 16:9 receiver is 210 horizontal cells by 75 vertical cells. The 4:3 coordinate-system grid size is 160 horizontal cells by 75 vertical cells. For all other aspect ratio formats, a percentage or relative positioning coordinate system may be used such that a grid is mapped to the screen with 100 horizontal cells and 100 vertical cells.

The grid coordinates are specified as a pair of values in the form: (horizontal, vertical). The origin is the point in the upper left-most corner of the safe-title area, and is assigned the coordinate (0, 0). For the 16:9 format, the upper-right corner is (209, 0), the lower-left corner is (0, 74), and the lower right corner is (209, 74). A similar set of reference points is defined for the 4:3 format's 160 x 75 coordinate system: respectively, (0,0), (159, 0), (0, 74), and (159, 74). It is important to remember that these grid cells are not intended for text positioning, but for window positioning.

Once the window is positioned, the starting positioning of the text rows and columns follow, depending upon the size of the displayed font. The ending of the text rows depends upon the spacing (monospace vs. proportional space) of the chosen font.

Figure 11 shows an exaggerated relationship between a 16:9 screen, the 210 x 75 grid, the overscan area, the viewable display area, the safe-title area, and the caption windows.

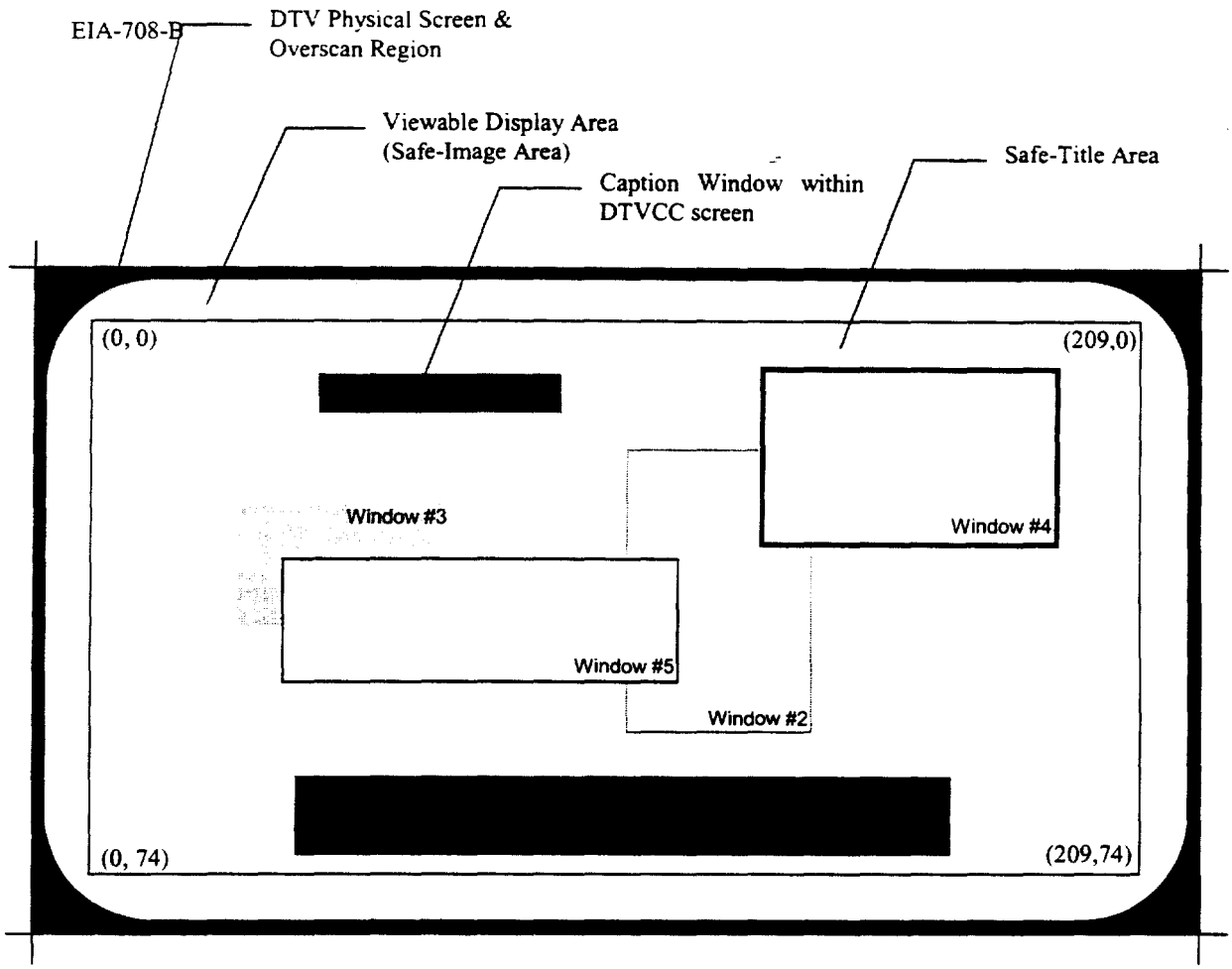


Figure 11 DTV 16:9 Screen and DTVCC Window Positioning Grid

8.3 User Options

Receiver manufacturers have the option to provide controls which may allow users to override styles and attributes specified in the service channel caption streams. Optional user controls might consist of caption font size, caption color and caption intensity (e.g., brightness) overrides. For further discussion, see the minimum DTVCC receiver decoder manufacturer recommendations in Section 9.

8.4 Caption Windows

All caption text is displayed and manipulated in receivers in the context of caption windows. There are 8 possible windows per service in which caption providers may write caption text. These windows may be implemented as buffers within receivers where any, none or all can be displayed at the same time. All 8 windows are available for the service currently selected by the user.

Manufacturers have the option of maintaining multiple sets of window buffers (i.e., instead of a single set of 8 buffers). Each set would be assigned to a service so that window processing of multiple services could occur simultaneously. This feature would have the effect that when a user switches services, the previously acquired and processed service data would be presented immediately. If only one set is used, the window buffers would have to be deallocated during service switching, and the new service would not appear until the buffers are reallocated and sufficient new service data are received.

The dimensions of a unique window specify an area on the screen which may contain caption text. The size of the window is based on the font size (SMALL, STANDARD, or LARGE) that the user has selected. Caption text designated for the window may not exceed the boundaries of the window, regardless of the font size the caption provider has specified or the font size the user has chosen.

A window's size may change on screen when a user changes the font size at the receiver. The effects of this window sizing are described further below.

8.4.1 Window Identifier

Each of the eight windows (and window buffers) is uniquely addressed by its window ID. Window ID numbers range from 0 to 7.

8.4.2 Window Priority

Each window has an associated priority which affects how it is displayed in conjunction with other windows that may be displayed at the same time. A higher priority (with 0 being the highest and 7 being the lowest) displayed window will overlap lower priority displayed windows on the screen.

8.4.3 Anchor Points

There are 9 locations within a window which serve as "anchors". An anchor specifies the reference point for positioning the window on the screen, and the "shrink and grow" directions (see Figure 12) of the window and caption text within the window when a user changes the font size.

8.4.4 Anchor ID

The 9 window anchor points are addressed by an Anchor ID which ranges from 0 to 8. Anchor ID 0 refers to the top-left corner of a window. Anchor 8 specifies the bottom-right corner of a window. Anchor 4 specifies the middle in the window.

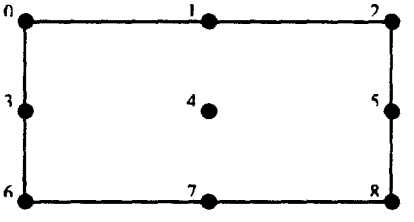


Figure 12 Anchor Points

Anchor points specify bounded and unbounded areas of caption text expansion and compression when the user overrides the standard font size for caption text display. Figure 13 shows the directions of expansion and compression of caption text for each anchor point. Solid lines indicate the bounded edges of the caption window. Dashed lines indicate the unbounded directions in which the caption text may shrink or grow.

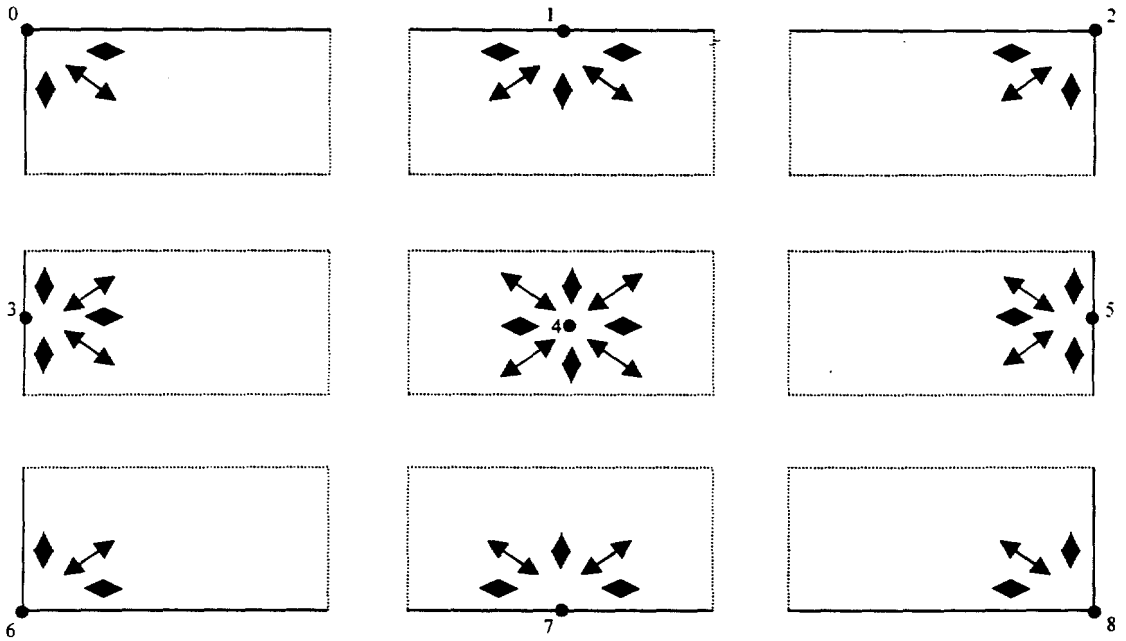


Figure 13 Implied Caption Text Expansion Based on Anchor Points

8.4.5 Anchor Location

The anchor location specifies where (in grid coordinates) the window's anchor point is to be physically located, and thus, the window itself. These grid coordinates are described in Section 8.2.

8.4.6 Window Size

The window size is specified in numbers of character rows and character columns for all display formats (16:9, 4:3, etc.). For all display formats, 15 is the maximum character row count and 32 is the maximum character column count.

> NOTE--A 16:9 format could handle longer rows (i.e., up to 42), but caption providers should keep the window column size at 32 characters, or less, in order to leave room on the display when the user selects the LARGE font size (see Section 8.4.7).

As for the physical size of the window on the screen, the receiver scales the window based on the "effective font size". The effective font size is based on a combination of the pen size chosen by the caption provider and the font size chosen by the receiver user. The height of the window is calculated as the number of rows multiplied by the physical height of the tallest character in the effective font size. The width of the window is calculated as the number of columns multiplied by the physical width of the widest character in the effective font size.

8.4.7 Window Row and Column Locking

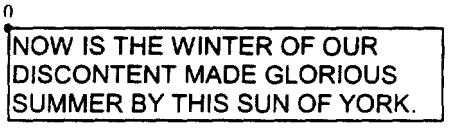
The "lock rows" and "lock columns" window parameters fix the maximum number of rows and columns of caption text that a window may have. If a row or column parameter is "unlocked", the receiver may automatically add or adjust columns or rows to a window under certain circumstances.

This means that in a text-mode or roll-up type caption service (with no embedded carriage returns) where the user has specified a font size smaller than intended by the caption provider, more rows and columns could fit into the physical window (as it appears on the screen) than specified in the original window size parameters (see examples below).

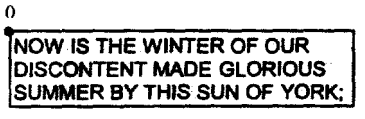
When the user has specified a font size larger than intended by the caption provider, the window's width may grow larger if the columns are locked. In order to insure that there is enough room on the display for 16:9 formats to grow a caption row so that it will fit on the screen, caption providers should not create windows or caption rows with more than 32 character columns. Since 16:9 formats can display 42 characters per row and if the widest character in the large font is no more than 1.3 (i.e., 42/32) times larger than the widest character in the standard font, the enlarged caption (and window) should fit on the screen without word wrapping.

8.4.7.1 Effects When Choosing Smaller Font

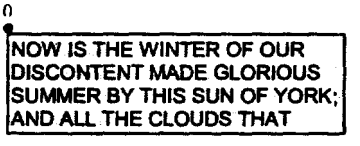
Figure 14 provides examples of the effects row and column locking have on a window and its STANDARD-sized caption text when a receiver user chooses the SMALL font.



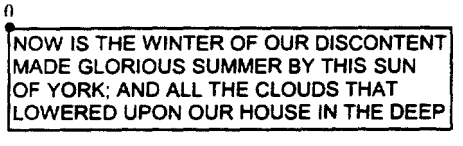
Original Caption : Standard Font Size in 3 row x 28 col window



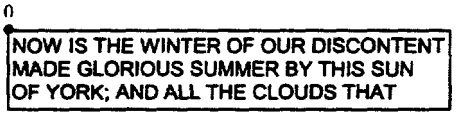
1. Shrink with Rows and Columns locked



2. Shrink with Rows unlocked and Columns locked



3. Shrink with Rows and Columns unlocked



4. Shrink with Rows locked and Columns unlocked

Figure 14 Examples of Caption Window Shrinking when User Picks Smaller Font

In the Figure 14 examples, the shaded rectangle indicates the size of the original window.

In example 1, the caption text and window shrink to a size in direct proportion to the original window. The caption contains the same number of lines and columns as the original, and the same words appear on the same line.

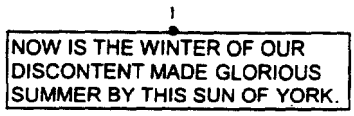
In example 2, the window indicates that there is room for another row of caption text. The window's height remains the same size.

In example 3, the window width and height remain the same; and with a smaller font, more characters may fit on a row. Thus, words are shifted up into the previous rows.

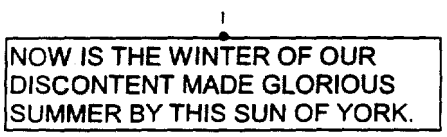
In example 4, the window shrinks in the vertical direction in order to maintain the same number of rows. The window height remains the same.

8.4.7.2 Effects When Choosing Larger Font

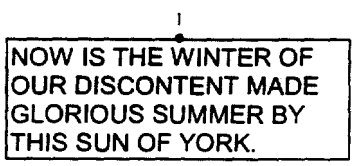
Figure 15 provides examples of the effects row and column locking have on a window and its STANDARD-sized caption text when a receiver user chooses the LARGE font.



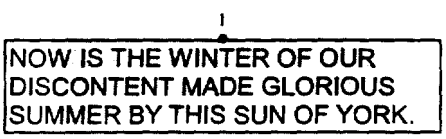
Original Caption : Standard Font Size in 3 row x 28 col window



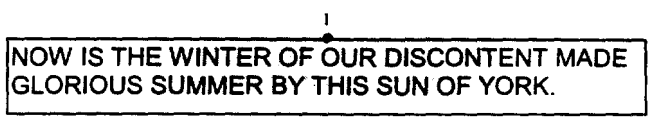
1. Grow with Rows and Columns locked



2. Grow with Rows unlocked and Columns locked



3. Grow with Rows and Columns unlocked



4. Grow with Rows locked and Columns unlocked

Figure 15 Examples of Caption Window Growing when Going to Larger Font

In the Figure 15 examples above, the shaded rectangle indicates the size of the original window.

In example 1, the caption text and window grow to a size in direct proportion to the original window. The caption contains the same number of lines and columns as the original, and the same words appear on the same line.

In example 2, the window width remains the same, but enough rows are automatically added to accommodate all of the caption text.

In example 3, the window is unrestricted in growth either in width or height. The decoder manufacturer is free to format the caption in anyway desired. However, since this allows for much ambiguity, it is suggested that the caption be formatted as in Example 1.

In example 4, the window height remains the same, but enough columns are automatically added to accommodate all of the caption text.

8.4.8 Word Wrapping

Whenever possible, the receiver should render embedded carriage returns as line breaks, since these carriage returns indicate an important aspect of the caption's formatting as determined by the service provider. However, it may sometimes be necessary for the receiver to ignore embedded line breaks. For example, if a caption is to appear in a larger font, and if its window's rows and/or columns are unlocked, the rows of text may need to become longer or shorter to fit within the allocated space. Such automatic reformatting of a caption is known as "word wrap."

The receiver should follow standard typographic practice when implementing word wrap. Potential breaking points (word-wrapping points) are indicated by the space character (20h) and by the hyphen character (2Dh).

If a row is to be broken at a space, the receiver should remove the space from the caption display. If a row is to be broken after a hyphen, the hyphen should be retained.

If an embedded return is to be removed, it should usually be replaced with a space. However, if the character to the left of the embedded return is a hyphen, the embedded return should be removed but NOT replaced with a space.

This specification does not include optional hyphens, nor does it provide for any form of automatic hyphenation. No non-breaking hyphen is defined. The non-breaking space (A0h in the G1 code set) and the non-breaking transparent space (21h in the G2 code set) should not be considered as potential line breaks.

If a single word exceeds the length of a row, the word should be placed at the start of a new row, broken at the character following the last character that fits on the row, and continued with further breaks if needed.

8.4.9 Window Text Painting

This section defines window text painting parameters and the interactions between them.

8.4.9.1 Justification

Caption text shall be formatted within the window based upon the justification type. The nomenclature for the justification types is based on standard English left-to-right Print Direction (see Section 8.4.9.2), and should be interpreted as follows:

Left Justification -- Text is justified to the start of the row, e.g., the left side for left-to-right print direction, the bottom for bottom-to-top print direction, and so on.

Right Justification -- The opposite of left justification. The text is aligned to the end of the row.

Centered Justification -- The text is centered horizontally when the print direction is left-to-right or right-to-left, and centered vertically when the print direction is top-to-bottom or bottom-to-top.

Full Justification -- The text is aligned to the left and right margins when the print direction is left-to-right or right-to-left, and to the top and bottom margins when the print direction is top-to-bottom or bottom-to-top.

8.4.9.2 Print Direction

The Print Direction parameter specifies in which direction characters will be written on a (horizontal or vertical) caption row (left-to-right, right-to-left, top-to-bottom, or bottom-to-top).

Table 16 defines how the cursor should move after drawing a character (when Justification is "Left" -- see Section 8.4.9.1) or after receiving a carriage return for each combination of Print Direction and Scroll Direction (see Section 8.4.9.3). Combinations of Print Direction and Scroll Direction that are not listed in this table are not permitted.

Print Direction	Scroll Direction	Cursor Movement	Carriage Return Behavior
Left -> Right	Top-> Bottom	increment column	decrement row, column=0
Left -> Right	Bottom->Top	increment column	increment row, column=0
Right->Left	Top-> Bottom	decrement column	decrement row, column=max
Right->Left	Bottom->Top	decrement column	increment row, column=max
Top-> Bottom	Left -> Right	increment row	decrement column, row=0
Top-> Bottom	Right->Left	increment row	increment column, row=0
Bottom->Top	Left -> Right	decrement row	decrement column, row=max
Bottom->Top	Right->Left	decrement row	increment column, row=max

Table 16 Cursor Movement After Drawing Characters

Figure 16 shows how a 3-row caption would appear using various justifications (see Section 8.4.9.1), Print Directions and Scroll Directions (see Section 8.4.9.3).

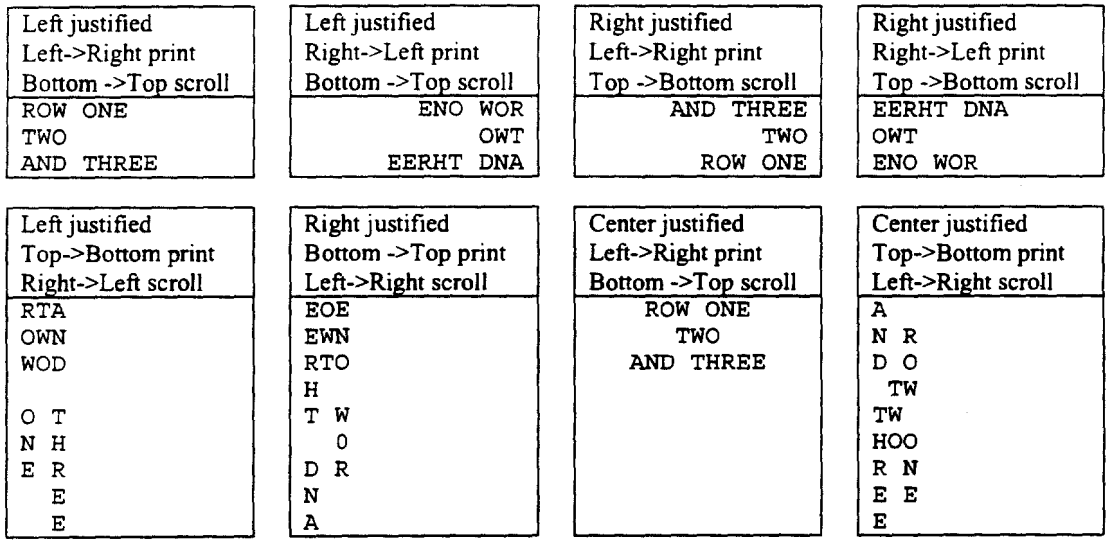


Figure 16 Examples of Various Justifications, Print Directions and Scroll Directions

8.4.9.3 Scroll Direction

The Scroll Direction parameter specifies in which direction the text will scroll (left-to-right, right-to-left, top-to-bottom, or bottom-to-top) when carriage returns and word wrapping is encountered. For example, NTSC style roll-up captions are achieved by specifying left-to-right printing with bottom-to-top scrolling.

Horizontal scrolling is allowed only with vertical print directions, and vertical scrolling with horizontal print directions (see Section 8.4.9.2). For example, left-to-right scrolling may be used with top-to-bottom or bottom-to-top printing, but not with left-to-right or right-to-left printing.

8.4.9.4 Combining Text Painting Attributes

This section describes one of a multitude of text painting effects using the various window attributes.

A 2-line ticker-tape effect can be accomplished with the following parameter settings: print direction set to "top-to-bottom" with scroll direction set to "right-to-left", and window size set to 2 rows and X columns. The initial pen location is set to the upper right corner of the window. One character from row 1 is sent, followed by 1 character from row 2, and a carriage return, and so on. The first character for row 1 will appear on the screen by itself. With top-to-bottom print direction, the 1st character for row 2 will appear beneath the first row 1 character. The carriage return is sent, creating a new column and causing the existing 2 characters to scroll to the left and the next row 1 character to appear to the right of the existing row 1 character.

8.4.10 Window Display

Caption text can be written to a window whether it is currently displayed (i.e., visible) or not. Writing a whole caption to a non-displayed window and then sending a **DisplayWindows** command will cause the caption to "pop-on" (if the SNAP display effect has been set for the window). To achieve successive "pop-on" type captions, two windows are required (just as displayed and non-displayed memory buffers are used in NTSC captioning), with the required sequencing of **DisplayWindows** and **HideWindows** commands to maintain the effect.

When a window is displayed or hidden, a Display Effect parameter can be specified which controls whether the window snaps on/off ("pop-on"), fades on/off, or wipes on/off. The rate of fade on/off may be a manufacturer's option, but an effect speed can be specified with the **SetWindowAttributes** command. The direction (left-to-right wipe, right-to-left wipe, top-to-bottom wipe, and bottom-to-top wipe) in which a window wipes on/off can also be specified with this command.

8.4.11 Window Colors and Borders

The area within a window can have different characteristics regarding color and opacity. Windows can be clear, or filled with a specified color. A window can be filled with an opaque color from a color palette (see Section 8.8). The colors within a window can have various effects associated with them. The window can be transparent (i.e., clear - that is, letting the underlying video show through), translucent (i.e., the underlying video is passed through a fixed level of color background filtering so that underlying video may partially show through the window), solid (opaque with a nominal level of saturation), or flashing (alternating transparency and opacity at the nominal, or solid, level of saturation).

Windows may have no borders, or be bounded by an enclosing border. Borders may be raised, depressed, uniform, or drop-shadowed (left or right). Manufacturers have a bit of latitude in how these border effects are achieved. A uniform border may appear as a single line surrounding the window. Raised, depressed and shadowed borders can be used to achieve 3-dimensional effects.

8.4.12 Predefined Window and Pen Styles

Colors, text painting, effects, and border types can be customized with the **SetWindowAttributes** and **SetPenAttributes** commands. However, the caption provider may wish to use predefined standard windows styles. A set of predefined styles will be hard stored in receivers. This set will anticipate the most widely used types of caption windows in order to conserve caption channel bandwidth by eliminating the need to transmit superfluous **SetWindowAttributes** and **SetPenAttributes** commands.

Predefined window and pen styles can be specified by the *window style* and *pen style* ID parameters in the **DefineWindow** command. See Section 9.12 for the defined "predefined window and pen styles".

8.5 Caption Pen

The caption text within a window is written with a *pen*. A pen governs the size, font, colors, and styles of the text within a window. Pen attributes are specified separately from window attributes through the use of the **SetPenAttributes** command. A window may contain text with more than one combination of pen attributes (i.e., different fonts, colors, etc.). Pen characteristics remain constant for a window unless specifically changed (via a subsequent **SetPenAttributes** command) by the caption provider. Pen attribute changes only affect text written after the change (i.e., the **SetPenAttributes** command does not change existing text within a window) and until a new **SetPenAttributes** command is encountered.

8.5.1 Pen Size

TV receivers may implement different sized characters (e.g., small, standard, and large) for supported fonts. Caption providers specify the pen size to be displayed, but users may override the size of the characters (i.e., which font size is to be viewed on the receiver) as desired.

8.5.2 Pen Spacing

Monospacing and proportional spacing are inherent attributes of the font chosen to write caption text. They cannot otherwise be controlled by either the caption provider or the user.

8.5.3 Font Styles

Caption providers may specify 1 of 8 different font styles to be used to write caption text. The styles specified in the "font style" parameter of the **SetPenAttributes** command are numbered from 0 through 7.

The following is a list of the 8 recommended font styles. For information purposes only, each font style references one or more popular fonts which embody the characteristics of the style:

0 - Default (undefined)
1 - Monospaced with serifs (similar to Courier)
2 - Proportionally spaced with serifs (similar to Times New Roman)
3 - Monospaced without serifs (similar to Helvetica Monospaced)
4 - Proportionally spaced without serifs (similar to Arial and Swiss)
5 - Casual font type (similar to Dom and Impress)
6 - Cursive font type (similar to Coronet and Marigold)
7 - Small capitals (similar to Engravers Gothic)

Implementation of these fonts styles by decoder manufacturers is optional. Font styles which are not supported in a decoder should be displayed in an available font which is most similar to the requested font style in the **SetPenAttributes** command.

8.5.4 Character Offsetting

Characters can be positioned relative to the row baseline in one of three ways: subscript (offset vertically downward), superscript (offset vertically upward), or normal (no offset).

8.5.5 Pen Styles

Pen styles can be italicized and/or underlined.

8.5.6 Foreground Color and Opacity

The foreground color and opacity of the caption characters can be specified by the caption provider. The character foreground opacity can be set to transparent (i.e., showing underlying video), translucent (i.e., showing a filtered level of underlying video), solid, or flashing.

8.5.7 Background Color and Opacity

Characters are individually contained within a small, rectangular background box. These background boxes have color and opacity attributes which may be specified separately from character foreground attributes.

8.5.8 Character Edges

The color attributes of the edges (or outlines) of the character foregrounds may be specified separately from the character foreground and background. Edge opacities have the same attribute as the character foreground opacities. The type of edge surrounding the body of the character may be NONE, RAISED, DEPRESSED, UNIFORM, or DROP_SHADOWED.

8.5.9 Caption Text Function Tags

Caption text can be tagged with a marker indicating the type of text content that is being encoded. This tagging is performed by caption providers via the **SetPenAttributes** command. With this command, caption text can be tagged with any one of the following qualities:

0 - Dialog (normal words being spoken by characters in the programming)
1 - Source or speaker ID (name of the speaker, or a description of the source of a sound)
2 - Electronically reproduced voice (spoken audio heard by the characters in the drama coming from a phone, radio, PA, etc.)
3 - Dialog in a language other than the drama's primary language
4 - Voiceover (narration or other disembodied voice NOT heard by the characters in the drama)
5 - Audible Translation (voice of a disembodied translator NOT heard by the characters in the drama)
6 - Subtitle Translation (text showing a translation into the primary language of the drama)
7 - Voice quality description (description of a voice quality)
8 - Song Lyrics (words being sung)
9 - Sound effect description (a description of a nonverbal sound or music heard by the characters in the drama)

10 - Musical score description (a description of background music NOT heard by the characters in the drama)
11 - Expletive (an interjectory word or expression, possible profane or harsh)
12 to 14 - (undefined)
15 - Text not to be displayed (reserved for future use by a text-based control and information channel within the caption text stream; e.g., hypertext, related non-caption program information)

In the previous list, all of the tagged text is to be displayed in the current caption window as indicated, except for tag 15 ("Text not to be displayed"), and optionally, tag 11 ("Expletive").

Decoder manufacturers have the flexibilty to display tagged text in an number of ways desired in order to enhance the caption viewing experience. For example, "Source or speaker ID" text may always be automatically displayed in italics by the decoder. This may be a feature which the viewer can enable and disable.

If no other information is available, decoders are to default caption text to the "Dialog" tag (0). That is, if caption text is received for a window when no **SetPenAttributes** command has been received, the text is to be treated as "Dialog".

8.6 Caption Text

Caption text is always written to the current caption window with the attributes set with the preceding **SetPenColor** and **SetPenAttributes** commands. There is no specific Text Write caption command; any displayable characters or codes from the G0, G1, G2, or G3 code sets in Service Blocks which are not part of any caption command are considered text to be written to the current window.

Caption text sequences must be terminated by either the start of a new DTVCC Command, or with an ASCII **ETX** (0x03) character when no other DTVCC Commands follow. This requirement aids decoders in processing text sequences when text spans multiple service blocks.

8.7 Caption Positioning

The starting row and column position of an individual character or set of characters may be specified at any time via the **SetPenLocation** command. The position of subsequent characters written after a location is specified depends upon the Print Direction and Scroll Direction specified for a window.

Character positions specify memory locations within the internal buffer holding the window text, and thus, do not specify physical locations on the screen. Screen locations are dependent on a multitude of pen and window attributes (e.g., character size, character spacing, justification, and print direction), and whether or not the font is proportionally spaced.

8.8 Color Representation

Foreground and background colors are specified in the Caption Commands as combinations of the red-green-blue color triad. Two bits are specified for each red, green, and blue color value which defines the intensity of each individual color component. Colors are specified as a group; i.e., (<red>, <green>, <blue>). The range of color specification is (0, 0, 0) [for black] through (3, 3, 3) [for bright white]. Bright red has a color value of (3, 0, 0); bright green has a color value of (0, 3, 0); and bright blue has a color value of (0, 0, 3). This coding scheme provides 64 different colors.

8.9 Service Synchronization

For the most part, caption providers insert caption commands and text into the DTVCC Caption Channel stream in a real-time manner. That is, captions are transmitted shortly before they are to be displayed. This is the technique used in NTSC captioning.

DTVCC provides for an additional synchronization capability by allowing DTVCC data to be pre-sent and processed at a later time, all under the control of the decoder. The **Delay** command provides this added functionality.

8.9.1 Delay Command

Decoders must maintain a Service Input Buffer for each of the services which can be processed simultaneously. This input buffer has a minimum size of 128 bytes. All DTVCC data for a service pass through the Service Input Buffer.

Most of the time, the data falls through the buffer instantaneously, and the data are processed by the DTVCC decoder as they are received.

The **Delay** command is used to instruct the decoder to suspend processing of the Service Input Buffer data for a specified time period. This command has a time-out period specified as its parameter. This period is specified in tenths of seconds.

When a **Delay** command is encountered, the decoder waits for the specified delay time to expire before processing any other service data. During the delay interval, incoming data for the active service is buffered in the Service Input Buffer. When the delay interval expires, interpretation of the incoming data is resumed.

The **Delay** command may be used in instances where the caption provider knows that it is about to lose access to the DTVCC encoder-to-decoder stream. Prior to losing access, the caption provider pre-sends a set of DTVCC commands with one or more embedded **Delay** commands. When the caption provider loses access, the decoders process the sequencing of the buffered data according to the providers' instructions.

Any active delay interval is automatically canceled when the Service Input Buffer becomes full. The decoder begins interpreting buffered data in order not to lose any incoming data.

8.9.2 DelayCancel Command

Caption providers may override an active **Delay** command via the **DelayCancel** command. The **DelayCancel** command terminates any currently pending Delay command processing, and causes the decoder to begin processing any data in the Service Input Buffer.

The **DelayCancel** command must be detected (recognized) prior to the input of Service Input Buffer. That is, the **DelayCommand** is not buffered. If it were, it would not be processed until the delay interval expires.

The **Delay** and **DelayCancel** commands are analogous to Service Input Buffer processing "suspend" and "resume". These two commands allow caption providers to preload a set of commands and with a **Delay** command (suspend) to delay their output (using the maximum delay interval), and then issue a **DelayCancel** command (resume) at a desired instance to have the commands executed en mass.

8.9.3 Reset Command

The **Reset** command reinitializes a DTVCC service. The effects of re-initialization are described in Section 8.9.5. The **Reset** command is encoded by a caption provider to reset caption service processing within decoders.

It is recommended that a **Reset** command be issued at the beginning of a captioned program, or program segment. This ensures that decoders are initialized and ready for a new program, and erases any left-over windows and captions that were not deleted as a result of such events as video up-cutting during transitions from one program source to another.

8.9.4 Reset and DelayCancel Command Recognition

In order for the **DelayCancel** and service **Reset** commands to be effective, they must be recognized after they are retrieved from the Caption Channel Data Stream and prior to their routing to (insertion into) the Service Input Buffer. All other commands are interpreted when they are pulled from the Service Input Buffer.

Figure 17 shows this point of "pre-processing" for a decoder which simultaneously processes multiple service streams.

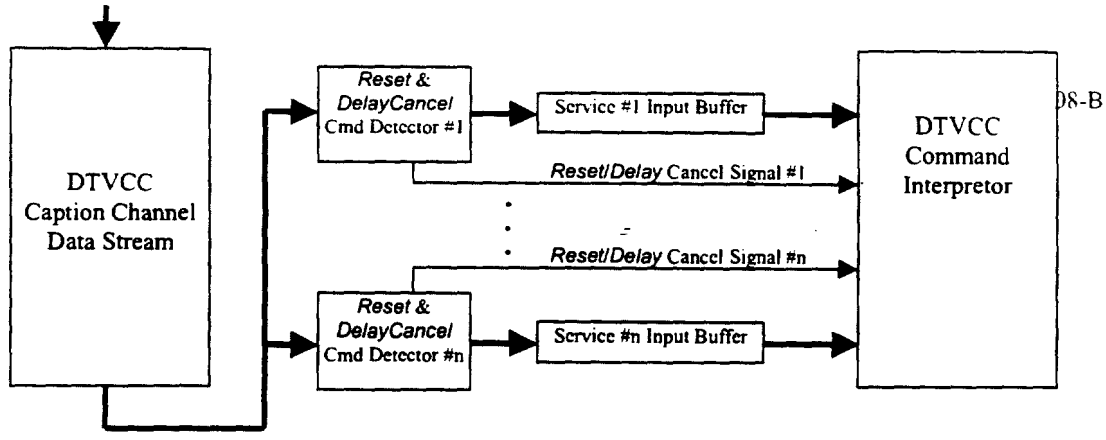


Figure 17 Reset & DelayCancel Command Detector(s) and Service Input Buffers

For descriptive purposes, Figure 18 shows a "hardware" implementation for the "*Reset & DelayCancel* Command Detector(s)"; a software implementation should be straight-forward. The Detectors may be implemented as a pair of 8-bit comparators. Since each of these commands is only a single byte in length (**Reset** = 0x8F, **DelayCancel** = 0x8E), this comparison may be simple, as is shown in Figure 18.

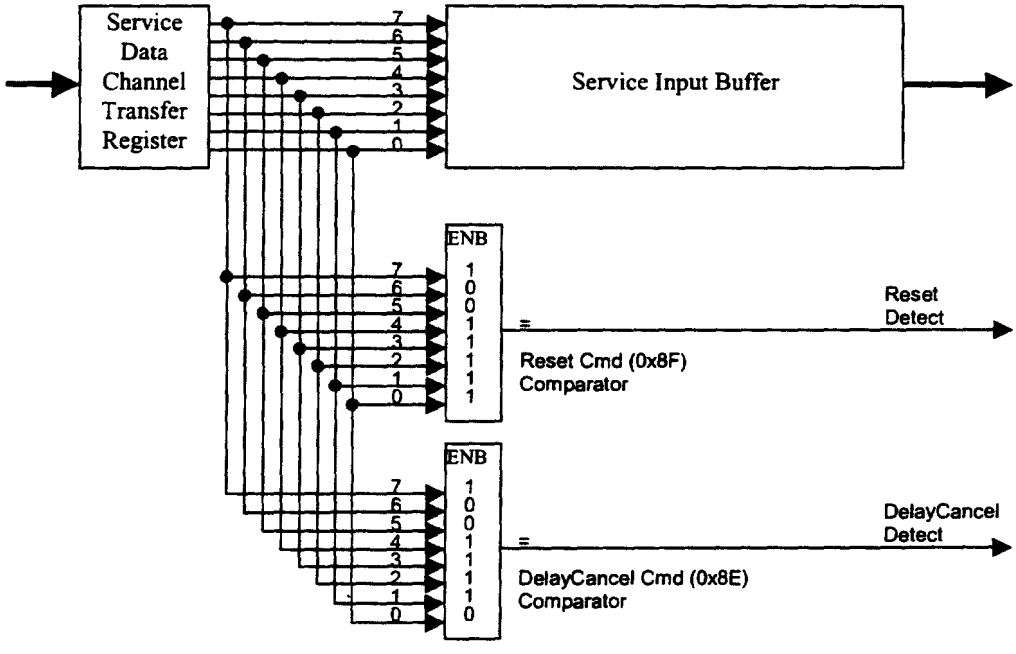


Figure 18 Reset & DelayCancel Command Detector(s) Detail

8.9.5 Service Reset Conditions

The "resetting" or "re-initialization" of a service means:

- that the service's windows are removed from the display,
- all defined windows for the service are deleted,
- all window and pen attributes for the service are deleted, and
- the Service Input Buffer is cleared.

A service shall be reset when any one of the following events occur:

1. a **Reset** command is received for a service.
2. a channel change occurs.
3. the Service Input Buffer overflows.
4. A loss of continuity in the Caption Channel Packet sequence_number.

8.10 DTVCC Command Set

This section presents the commands which may be encoded by caption providers. The commands are grouped into the following command types: Window Commands, Pen Commands, Caption Text Commands, and Synchronization Commands.

8.10.1 Window Commands

These commands create, delete, modify, and display windows, and specify the current caption window for a caption service.

Command Code	Command Name	Parameters
CW0, ... , CW7	**SetCurrentWindow**	window ID
DF0, ... , DF7	**DefineWindow**	window ID, priority, anchor point, relative positioning, anchor vertical, anchor horizontal, row count, column count, row lock, column lock, visible, window style ID, pen style ID
DLW	**DeleteWindows**	window map
DSW	**DisplayWindows**	window map
HDW	**HideWindows**	window map
TGW	**ToggleWindows**	window map
SWA	**SetWindowAttributes**	justify, print direction, scroll direction, wordwrap, display effect, effect direction, effect speed, fill color, fill opacity, border type, border color

8.10.2 Pen Commands

These commands define pen attributes and colors.

Command Code	Command Name	Parameters
SPA	**SetPenAttributes**	pen size, font, text tag, offset, italics, underline, edge type
SPC	**SetPenColor**	fg color, fg opacity, bg color, bg opacity, edge color
SPL	**SetPenLocation**	row, <column

8.10.3 Synchronization Commands

These commands control the rate of service data interpretation.

Command Code	Command Name	Parameters
DLY	**Delay**	tenths of seconds
DLC	**DelayCancel**	
RST	**Reset**	

8.10.4 Caption Text

There is no specific Text Write caption command. That is, any characters or codes from the G0, G1, G2, or G3 code sets in Service Blocks which are not part of any caption command are considered text to be written to the current window. Any such encountered text is written to the current window starting at the window's current cursor position. The window's current cursor is adjusted automatically to the row and column following the last character in the text string.

In order to assist decoders in determining the end of caption text segments for dangling segments at the end of a series of caption commands and text, an ETX code (0x03) from the C0 Code Space is to be inserted at the end of the text segment to terminate the segment. Text which is immediately followed by a caption command code (from the C1 Code Space) does not require the ETX code. That is, the decoder determines the end of a text segment when it encounters a caption command code, or an ETX code.

Without the ETX assist, decoding software may find it difficult to know if it should wait until the next service block to see if there is more caption text for a caption row when a service block terminates in text. If the last used byte in the block is an ETX, then the decoder knows there is no more text for this segment in a subsequent service block, and the decoder can process it as a single entity (this is especially helpful when doing Center justification of a row and word wrap processing, for example). If the last used byte in the block is a text character, then the decoder must wait until the next service block to see if there is any more text.

8.10.5 Command Descriptions

Each command is described in detail within this section.

SET CURRENT WINDOW - (CWx)

Name: **SetCurrentWindow** - Specify current window ID

Command Type: Window

Format: **SetCurrentWindow** (*window ID*)

Parameters: ■ *window ID* (id) is the unique window identifier (0 - 7).

Command Coding: **CW0, ... , CW7 = 80h, ... , 87h (10000000b, ... , 10000111b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	0	id_2	id_1	id_0	command

Description: **SetCurrentWindow** specifies the window to which all subsequent window style and pen/text commands are directed. The *window ID* must address a window which has already been created by the **DefineWindow** command. This command initializes a "current window ID" variable internal to the decoder. SetCurrentWindow directs the following commands to the specified window: **SetWindowAttributes, SetPenAttributes, SetPenColor, SetPenLocation, WriteText**.

DEFINE WINDOW - (DF0 ... DF7)

Name: **DefineWindow** - Create window and set initial parameters

Command Type: Window

Format: **DefineWindow** (*window ID, priority, anchor point, relative positioning, anchor vertical, anchor horizontal, row count, column count, row lock, column lock, visible, window style ID, pen style ID)*

Parameters:

- *window ID* (id) is the unique window identifier (0 - 7)
- *priority* (p) is the window display priority (0 - 7)
- *anchor point* (ap) is the window anchor position number to use for the window's position on the screen (0 - 8).
- *relative positioning* (rp) is a flag that, when set to 1, indicates that the *anchor vertical* (av) and *anchor horizontal* (ah) coordinates specify "relative coordinates" (i.e., percentages) instead of physical screen coordinates.
- *anchor vertical* (av) is the vertical position of the window's anchor point on the viewing screen when the window is displayed (0 – 74 for 16:9 and 4:3 systems and the *relative positioning* (rp) parameter is set to 0, or 0-99 when the 'rp' parameter is set to 1).
- *anchor horizontal* (ah) is the horizontal position of the window's anchor point on the viewing screen when the window is displayed (0 - 209 for 16:9 systems, 0 - 159 for 4:3 systems and the *relative positioning* (rp) parameter is set to 0, or 0-99 when the 'rp' parameter is set to 1).
- *row count* (rc) is the number of virtual rows of text (assuming the STANDARD *pen size;* see **SetPenAttributes**) the window body will hold (0 - 11).
- *column count* (cc) is the number of virtual columns of text (assuming the STANDARD *pen size;* see **SetPenAttributes**) the window body will hold (0 - 31 for 4x3 formats, and 0 - 41 for 16x9 formats).
- *row lock* (rl), when set to YES, fixes the absolute number of rows of caption text the window will contain. When NO, *row lock* permits a receiver to add more rows to a window if the user selects a smaller size font other than intended by the caption provider. [YES, NO] = [1, 0].
- *column lock* (cl), when set to YES, fixes the absolute number of columns of caption text the window will contain. When NO, *column lock* permits a receiver to add more columns to a window if the user selects a smaller size font other than intended by the caption provider. [YES, NO] = [1, 0].
- *visible* (v), when set to YES, causes the window to be viewed (i.e., displayed) on the screen immediately after it is created. When set to NO, the window is not displayed (i.e., hidden) after it is created. [YES, NO] = [1, 0].
- *window style ID* (ws), when non-zero, specifies 1 of 7 static preset window attribute styles to use for the window when it is created (0 - 7). When zero during a window create, the window style is automatically set to window style #1. When zero during a window update, no window attribute parameters are changed. See **SetWindowAttributes** command.
- *pen style ID* (ps), when non-zero, specifies 1 of 7 static preset pen attribute styles to use for the window when it is created (0 - 7). When zero during a window create, the pen style is automatically set to pen style #1. When zero during a window update, no pen attribute parameters are changed. See **SetPenAttributes** command.

Command Coding: **DF0, ... , DF7 = 98h, ... , 9Fh (10011000b, ... , 10011111b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	**0**	**0**	**1**	**1**	id_2	id_1	id_0	command
0	**0**	v	rl	cl	p_2	p_1	p_0	parm1
rp	av_7	av_6	av_5	av_4	av_3	av_1	av_0	parm2
ah_7	ah_6	ah_5	ah_4	ah_3	ah_2	ah_1	ah_0	parm3
ap_3	ap_2	ap_1	ap_0	rc_3	rc_2	rc_1	rc_0	parm4
0	**0**	cc_5	cc_4	cc_3	cc_2	cc_1	cc_0	parm5
0	**0**	ws_2	ws_1	ws_0	ps_2	ps_1	ps_0	parm6

Description:

DefineWindow creates a window for the specified window identifier (*window ID*) and initializes the window with the non-style parameters listed in the command and with the available static style presets specified with the *window style ID* and *pen style ID* parameters. If the window is not already defined when the **DefineWindow** command is received, the window is created; otherwise it is simply updated. If the window is being created, all character positions in the window are set to the window fill color and the pen location is set to (0, 0). The **DefineWindow** command also makes the defined window the current window (see **SetCurrentWindow**).

When a window is created, a specific or automatic *window style ID* is assigned in order to preload a set of known window attribute values. These attributes can be subsequently modified with the **SetWindowAttributes** command. A *pen style ID* is assigned in the same fashion. Pen style attributes can be subsequently modified with the **SetPenAttributes** command.

When a decoder receives a **DefineWindow** command for an existing window, the command is to be ignored if the command parameters are unchanged from the previous window definition. Encoders or caption providers may periodically repeat window definitions in order for receivers to acquire a service and begin decoding and displaying captions with little delay. It is suggested that all window and pen definition and attribute commands be repeated in this way.

When an existing window is being updated (e.g., resized or moved) with the **DefineWindow** command, the pen location and pen attributes are unaffected.

CLEAR WINDOWS - (CLW)

Name: **ClearWindows** - Clears Text from a set of windows

Command Type: Window

Format: **ClearWindows** (*window map*)

Parameters:

- *window map* (w) is an 8-bit bitmap which specifies the window(s) affected by the command. Each bit position represents a window (i.e., *window ID*) to be affected (e.g., bit position 4 addresses the window with the *window ID* of 4). A value of 1 in a bit position specifies that the associated window is to be processed by the command. A bit value of 0 indicates that the associated window is unaffected by the command.

Command Coding: **CLW = 88h (10001000b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	**0**	**0**	**0**	**1**	**0**	**0**	**0**	command
w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	parm1

Description: **ClearWindows** removes any existing text from the specified window(s) in the window map. When a window is cleared, the entire window is filled with the window fill color.

DELETE WINDOWS - (DLW)

Name: **DeleteWindows** - Deletes window definitions for a set of windows

Command Type: Window

Format: **DeleteWindows** (*window map*)

Parameters:

- *window map* (w) is an 8-bit bitmap which specifies the window(s) affected by the command. Each bit position represents a window (i.e., *window ID*) to be affected (e.g., bit position 4 addresses the window with the *window ID* of 4). A value of 1 in a bit position specifies that the associated window is to be processed by the command. A bit value of 0 indicates that the associated window is unaffected by the command.

Command Coding: **DLW = 8Ch (10001100b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	1	0	0	command
w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	parm1

Description: **DeleteWindows** removes all specified windows from the receiver. For example, a *window map* value of 64 (hex) deletes windows 6 (w_6), 5 (w_5), and 2 (w_2). A *window map* value of FF (hex) deletes all defined windows in the receiver. If the current window is deleted, then the decoder's current window ID is unknown and must be reinitialized with either the **SetCurrentWindow** or **DefineWindow** command.

DISPLAY WINDOWS - (DSW)

Name: **DisplayWindows** – Causes a set of windows to become visible

Command Type: Window

Format: **DisplayWindows** (*window map*)

Parameters:

■ *window map* (w) is an 8-bit bitmap which specifies the window(s) affected by the command. Each bit position represents a window (i.e., *window ID*) to be affected (e.g., bit position 4 addresses the window with the *window ID* of 4). A value of 1 in a bit position specifies that the associated window is to be processed by the command. A value of 0 indicates that the associated window is unaffected by the command..

Command Coding: **DSW = 89h (10001001b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	0	0	1	command
w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	parm1

Description: **DisplayWindows** causes the specified, existing windows to be visible on the receiver display screen. For example, a *window map* value of 96 (hex) displays windows 7 (w_7), 4 (w_4), 2 (w_2), and 1 (w_1). A *window map* value of FF (hex) causes all existing windows in the receiver to be displayed. This command does not affect the current window ID.

HIDE WINDOWS - (HDW)

Name: **HideWindows** – Causes a set of windows to become invisible

Command Type: Window

Format: **HideWindows** (*window map*)

Parameters:

■ *window map* is an 8-bit bitmap which specifies the window(s) affected by the command. Each bit position represents the window (i.e., *window ID*) to be affected (e.g., bit position 4 addresses the window with the *window ID* of 4). A value of 1 in a bit position specifies that the associated window is to be processed by the command. A value of 0 indicates that the associated window is unaffected by the command.

Command Coding: **HDW = 8Ah (10001010b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	0	1	0	command
w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	parm1

Description: **HideWindows** causes all specified and currently defined windows to be removed from the receiver display screen. For example, a *window map* value of 72 (hex) hides windows 6 (w_6), 5 (w_5), 4 (w_4), and 1 (w_1). A *window map* value of FF (hex) causes all existing windows in the receiver to be hidden.

TOGGLE WINDOWS - (TGW)

Name: **Toggle Windows** - Toggles Display/Hide status of a set of windows

Command Type: Window

Format: **Toggle Windows** (*window map*)

Parameters: ■ *window map* is an 8-bit bitmap which specifies the window(s) affected by the command. Each bit position represents the window (i.e., *window ID*) to be affected (e.g., bit position 4 addresses the window with the *window ID* of 4). A value of 1 in a bit position specifies that the associated window is to be processed by the command. A value of 0 indicates that the associated window is unaffected by the command.

Command Coding: **TGW = 8Bh (10001011b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	**0**	**0**	**0**	**1**	**0**	**1**	**1**	command
w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	parm1

Description: **Toggle Windows** causes all specified and currently defined windows to toggle their display/hide status. That is, the specified windows in the *window map* which are currently displayed will be hidden, and the hidden ones will be displayed. For example, a *window map* value of 83 (hex) toggles windows 7 (w_7), 1 (w_1), and 0 (w_0). A *window map* value of FF (hex) causes all existing windows in the receiver to be toggled.

SET WINDOW ATTRIBUTES - (SWA)

Name: **SetWindowAttributes** - Defines the window styles for the current window.

Command Type: Window

Format: **SetWindowAttributes** (*justify, print direction, scroll direction, wordwrap, display effect, effect direction, effect speed, fill color, fill opacity, border type, border color)*

Parameters:

- *justify* (j) specifies how the text to be written in the window will be justified. [LEFT, RIGHT, CENTER, FULL] == [0, 1, 2, 3].
- *print direction* (pd) specifies in which direction text will be written in the window. [LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP] == [0, 1, 2, 3].
- *scroll direction* (sd) specifies which direction text will scroll when the end of a caption "line" is reached. [LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP] == [0, 1, 2, 3]..
- *wordwrap* (ww), when set to YES, word wrapping is enabled. When set to NO, wordwrapping is disabled. [YES, NO] == [1, 0].
- *display effect* (de) specifies the effect that is to take place when the window is displayed and when it is hidden. When the SNAP effect is chosen, the window will pop-on the screen when the window is displayed and pop-off when the window is hidden. The FADE effect causes the window to fade onto and off of the screen at the specified *effect rate*. The WIPE effect causes the window to swipe onto and off of the screen at the specified *effect rate* and *effect direction*. [SNAP, FADE, WIPE] == [0, 1, 2].
- *effect direction* (ed) specifies which direction a WIPE window will appear on the screen. Note that a WIPE window will wipe-off of the screen in the opposite direction from which it wiped-on. [LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM, BOTTOM_TO_TOP] == [0, 1, 2, 3].
- *effect speed* (es) specifies, in .5 second units, how fast windows with WIPE and FADE effects will appear and disappear from the screen when they are displayed and hidden. The effect speed value may range from 1 to 15, approximating .5 (1 x .5) to 7.5 (15 x .5) seconds of effect speed variation.
- *fill color* (fr, fg, fb) is the color of the windows interior (see Section 8.4.11).
- *fill opacity* (fo) is the characteristic of the fill color of the window and the window border. [SOLID, FLASH, TRANSLUCENT, TRANSPARENT] == [0, 1, 2, 3].
- *border type* (bt) defines the type of outer edge surrounding the window. [NONE, RAISED, DEPRESSED, UNIFORM, SHADOW_LEFT, SHADOW_RIGHT] == [0, 1, 2, 3, 4, 5].
- *border color* (br, bg, bb) is the color of the windows outer edge(see Section 8.4.11).

Command Coding: **SWA = 97h (10010111b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	**0**	**0**	**1**	**0**	**1**	**1**	**1**	command
fo_1	fo_0	fr_1	fr_0	fg_1	fg_0	fb_1	fb_0	parm1
bt_1	bt_0	br_1	br_0	bg_1	bg_0	bb_1	bb_0	parm2
bt_2	ww	pd_1	pd_0	sd_1	sd_0	j_1	j_0	parm3
es_3	es_2	es_1	es_0	ed_1	ed_0	de_1	de_0	parm4

Description: **SetWindowAttributes** assigns the specified style attributes to the current window. This command can be issued any number of times to an existing window. The style attributes will overwrite any existing attributes assigned to the window.

SET PEN ATTRIBUTES - (SPA)

Name: **SetPenAttributes** - Assign pen style attributes for the current window.

Command Type: Pen

Format: **SetPenAttributes** (*pen size, font, text tag, offset, italics, underline, edge type)*

Parameters:

■ *pen size* (s) defines which of three pen sizes is to be used for text written to the current window, as specified by the current window ID. Note that the pen size displayed on the screen can be overridden by the user. [SMALL, STANDARD, LARGE] == [0, 1, 2].

■ *font style* (fs) specifies which one of 8 different predefined font styles (see Section 8.5.3) to be used for text written to the current window (0 - 7).

0 - Default (undefined)
1 - Monospaced with serifs
2 - Proportionally spaced with serifs
3 - Monospaced without serifs
4 - Proportionally spaced without serifs
5 - Casual font type
6 - Cursive font type
7 - Small capitals

■ *text tag* (tt) specifies which one of 16 different predefined caption text function tags (see Section 8.5.9) is to be associated with the following caption text to be written to the current window (0 - 15).

0 - Dialog
1 - Source or speaker ID
2 - Electronically reproduced voice
3 - Dialog in language other than primary
4 - Voiceover
5 - Audible Translation
6 - Subtitle Translation
7 - Voice quality description
8 - Song Lyrics
9 - Sound effect description
10 - Musical score description
11 - Expletive
12 to 14 - (undefined)
15 - Text not to be displayed

■ *offset* (o) specifies sub-scripting and super-scripting attributes for text written to the current window. [SUBSCRIPT, NORMAL, SUPERSCRIPT] == [0, 1, 2].

■ *italics* (i) specifies if text written to the current window is italicized. [YES, NO] == [1, 0].

■ *underline* (u) specifies if text written to the current window is underlined. [YES, NO] == [1, 0].

■ *edge type* (et) is the type of outlined edge of the text. [NONE, RAISED, DEPRESSED, UNIFORM, LEFT_DROP_SHADOW, RIGHT_DROP_SHADOW] == [0, 1, 2, 3, 4, 5].

Command Coding: **SPA = 90h (10010000b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	**0**	**0**	**1**	**0**	**0**	**0**	**0**	command
tt_3	tt_2	tt_1	tt_0	o_1	o_0	s_1	s_0	parm1
i	u	et_2	et_1	et_0	fs_2	ft_1	ft_0	

Description: **SetPenAttributes** assigns pen style attributes for the currently defined window. Text written to the current window will have the attributes specified by the most recent **SetPenAttributes** command written to the window. Pen attributes for a window can be changed as often as desired. These attributes will remain in effect for the window during its entire existence.

SET PEN COLOR - (SPC)

Name: **SetPenColor** - Assign styles to a dynamic preset style number.

Command Type: Pen

Format: **Set Pen Color** (*fg color, fg opacity, bg color, bg opacity, edge color)*

Parameters:

- *fg color* (fr, fg, fb) is the color of the text foreground body (see Section 8.5.6).
- *fg opacity* (fo) is the characteristic of the text foreground body. [SOLID, FLASH, TRANSLUCENT, TRANSPARENT] == [0, 1, 2, 3].
- *bg color* (br, bg, bb) is the color of the background box surrounding the window text (see Section 8.5.7).
- *bg opacity* (bo) is the characteristic of the text background. [SOLID, FLASH, TRANSLUCENT, TRANSPARENT] == [0, 1, 2, 3].
- *edge color* (er, eg, eb) is the color of the outlined edges of the text. The text character edges have the same opacity value as *fg opacity* (see Section 8.5.6).

Command Coding: **SPC = 91h (10010001b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	1	0	0	0	1	command
fo_1	fo_0	fr_1	fr_0	fg_1	fg_0	fb_1	fb_0	parm1
bo_1	bo_0	br_1	br_0	bg_1	bg_0	bb_1	bb_0	parm2
0	0	er_1	er_0	eg_1	eg_0	eb_1	eb_0	parm3

Description: **SetPenColor** assigns the pen color attributes for the current window, as specified by the current window ID. Text written to the current window will have the color attributes specified by the most recent **SetPenColor** command written to the window. Pen color attributes for a window can be changed as often as desired. These attributes will remain in effect for the window during its entire existence.

SET PEN LOCATION - (SPL)

Name: **SetPenLocation - Specifies the pen cursor location within a window.**

Command Type: Pen

Format: **SetPenLocation** *(row, column)*

Parameters:
- *row* (r) is the text row within the current window's text buffer (0-14).
- *column* (c) is the text column within the current window's text buffer (0-31 for 4:3 formats, 0-41 for 16:9 formats).

Command Coding: **SPL = 92h (10010010b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	1	0	0	1	0	command
0	0	0	0	r_3	r_2	r_1	r_0	parm1
0	0	c_5	c_4	c_3	c_2	c_1	c_0	parm2

Description: **SetPenLocation** repositions the pen cursor for the current window, as specified by the current window ID. . When the window justification type is "left," The next group of text written to the current window will start at the specified row and column, and justification will be ignored. When the window justification type is not left and the print direction is left-to-right or right-to-left, the column parameter shall be ignored. When the window justification type is not left and the print direction is top-to-bottom or bottom-to-top, the row parameter shall be ignored. When the window justification type is not left, text shall be formatted based upon the current window justification type. Note that if a window is not *locked* (see **DefineWindow**) and the SMALL *pen size* (see **SetPenAttributes**) is in effect, more than 12 rows and 36 columns could possibly be addressed.

DELAY - (DLY)

Name: **Delay** - Delays service data interpretation

Command Type: Synchronization

Format: **Delay** *(tenths of seconds)*

Parameters: ■ *tenths of seconds* (t) is the number of tenths of seconds to delay before recommencing service data interpretation.

Command Coding: **DLY = 8Dh (10001101b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	1	0	1	command
t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0	parm1

Description: **Delay** instructs receivers to suspend interpretation of the current service's command input buffer. The delay is specified in tenths of seconds. Once the delay time expires, interpretation of caption commands recommences.

The delay value may range from 1 to 255 -- which specifies an effective delay time from 1/10 to 25.5 (255/10) seconds.

A delay for a service will remaining in effect until one of the following occurrences:

- the specified delay time expires
- a DelayCancel command is received
- the service's input buffer becomes full
- a service Reset command is received

DELAY CANCEL - (DLC)

Name: **DelayCancel** - Cancels an Active-Delay Command

Command Type: Synchronization

Format: **DelayCancel**

Parameters: none

Command Coding: **DLC = 8Eh (10001110b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	1	1	0	command

Description: **DelayCancel** command terminates any active **Delay** command processing within the decoder.

RESET - (RST)

Name: **Reset** - Resets the Caption Channel Service

Command Type: Synchronization

Format: **Reset**

Parameters: none

Command Coding: **RST = 8Fh (10001111b)**

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	
1	0	0	0	1	1	1	1	command

Description: **Reset** command reinitializes the service for which it is received.